

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日
Date of Application: 2 0 0 2 年 1 2 月 9 日

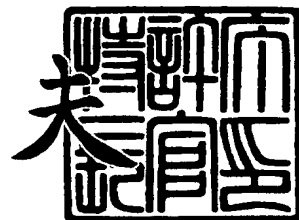
出 願 番 号
Application Number: 特 願 2 0 0 2 - 3 5 6 9 7 5
[ST. 10/C]: [J P 2 0 0 2 - 3 5 6 9 7 5]

出 願 人
Applicant(s): シャープ株式会社

2 0 0 3 年 1 1 月 7 日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康 夫



【書類名】 特許願

【整理番号】 186290

【提出日】 平成14年12月 9日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 7/00
G06F 11/00

【発明者】

【住所又は居所】 大阪府大阪市阿倍野区長池町 2 2 番 2 2 号 シャープ株式会社内

【氏名】 牧田 廣一

【発明者】

【住所又は居所】 大阪府大阪市阿倍野区長池町 2 2 番 2 2 号 シャープ株式会社内

【氏名】 大西 充久

【特許出願人】

【識別番号】 000005049

【住所又は居所】 大阪府大阪市阿倍野区長池町 2 2 番 2 2 号

【氏名又は名称】 シャープ株式会社

【代理人】

【識別番号】 100062144

【弁理士】

【氏名又は名称】 青山 葆

【選任した代理人】

【識別番号】 100086405

【弁理士】

【氏名又は名称】 河宮 治

【選任した代理人】

【識別番号】 100084146

【弁理士】

【氏名又は名称】 山崎 宏

【選任した代理人】

【識別番号】 100122286

【弁理士】

【氏名又は名称】 仲倉 幸典

【手数料の表示】

【予納台帳番号】 013262

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 0208766

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 デバッグ装置、デバッグ方法および記録媒体

【特許請求の範囲】

【請求項 1】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段とを備えたことを特徴とするデバッグ装置。

【請求項 2】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とするデバッグ装置。

【請求項 3】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とするデバッグ装置。

【請求項 4】 請求項 1 乃至 3 のいずれか一つに記載のデバッグ装置において、

上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応を表すことを特徴とするデバッグ装置。

【請求項 5】 請求項 1 乃至 3 のいずれか一つに記載のデバッグ装置におい

て、

上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の変数名の対応を表すことを特徴とするデバッグ装置。

【請求項 6】 請求項 1 乃至 3 のいずれか一つに記載のデバッグ装置において、

上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応と変数名の対応とを表すことを特徴とするデバッグ装置。

【請求項 7】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、
上記デバッグ情報を記憶手段に記憶させることを特徴とするデバッグ方法。

【請求項 8】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とするデバッグ方法。

【請求項 9】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、
上記デバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とするデバッグ方法。

【請求項 10】 請求項 7 乃至 9 のいずれか一つに記載のデバッグ方法を実現させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

この発明は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置およびデバッグ方法に関する。また、そのようなデバッグ方法を実現させるためのプログラムを記録した記録媒体に関する。

【0002】

なお、並列プログラミング言語により記述されたプログラムを並列プログラムと呼ぶ。

【0003】

また、逐次処理を記述するためのプログラミング言語を逐次プログラミング言語、それにより記述されたプログラムを逐次プログラムと呼ぶ。

【0004】

また、デバッグとはプログラムの不具合を特定して修正することを意味する。

【0005】

【従来の技術】

一般に、並列にいくつかの動作を同時に行う処理を記述する場合、並列プログラミング言語が用いられる。例えば、大規模集積回路（LSI）の設計においては、その内部の動作は基本的に並列動作であるため、プログラミング言語としては、逐次処理を記述するプログラミング言語ではなく、並列プログラミング言語が用いられる。この並列プログラミング言語により設計されたLSIの動作を検証するためには、本来、そのLSIの動作が記述された並列プログラムを計算機上で動作させるのが望ましい。

【0006】

しかし、LSIの設計者が設計のために使用している計算機は、逐次プログラムを実行するものであることが多い。そのような逐次プログラムを実行する計算機を使って、並列プログラムを直接実行することはできない。

【0007】

そこで、図1に示すように、並列プログラム1を逐次化装置2によって自動的

に逐次プログラム 4 に変換し、得られた逐次プログラム 4 をその計算機上で実行する方法が広く行われている。逐次プログラム 4 を実行した結果、元の並列プログラム 1 にバグ（不具合）があることが分かった場合、設計者は逐次プログラム 4 をデバッガ 6 上で動作させて不具合個所を特定し、その後、元の並列プログラム 1 の対応する個所を修正する。

【0008】

【発明が解決しようとする課題】

ところで、並列プログラム 1 を逐次化装置 2 によって自動的に逐次プログラム 4 に変換する際に、並列プログラム 1 での一つの行が、逐次プログラム 4 では複数の行からなる処理に書換えられる場合がある。また、変数名についても、並列プログラム 1 での或る変数名が、逐次プログラム 4 では別の変数名に書き換えられる場合がある。

【0009】

しかし、上述の方法では、並列プログラム 1 から逐次プログラム 4 への変換時には、並列プログラム 1 での或る行が逐次プログラム 4 ではどの行に書換えられたかや、並列プログラム 1 での或る変数名が逐次プログラム 4 ではどの変数名に書換えられたかの情報が残らない。

【0010】

そのため、デバッグを行う作業者は、並列プログラム 1 での行と逐次プログラム 4 での行との対応を自身で取りながら作業を行わなければならない。また、並列プログラムでの変数名と逐次プログラムでの変数名との対応も自身で取りながら作業を行わなければならない。このため、並列プログラムを逐次プログラムに変換してデバッグを行う作業は、非常に効率が悪くなり、時間がかかるという問題がある。

【0011】

また、図 2 に示すように、並列プログラム 1 を一旦逐次プログラム 4 に変換し、逐次プログラム 4 に不具合があった場合は、元の並列プログラム 1 ではなく逐次プログラム 4 を修正し、修正後の逐次プログラムを並列化装置 9 によって並列プログラム 10 に変換する方法も知られている（例えば、特許文献 1 参照。）。

【0012】

しかし、この方法によっても、作業者は、並列プログラム1におけるのとは異なる行や変数名を含む逐次プログラム4をデバッグしなければならない。このためデバッグ作業の効率が悪くなる。また、この方法によれば、逐次プログラム4をデバッグした後に並列プログラム10に変換するため、得られた並列プログラム10が、元の並列プログラム1の作成者が意図しない構造に変更されてしまう可能性がある。そのため、プログラムの分り易さが低下し、後で、並列プログラム10を使用または変更しようとする際の作業効率がかかる。

【0013】**【特許文献1】**

特開平8-16429号公報

【0014】

そこで、この発明の課題は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、作業者がデバッグ作業を効率良く行うことができるものを提供することにある。

【0015】

また、この発明の課題は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、作業者がデバッグ作業を効率良く行うことができるものを提供することにある。

【0016】

また、この発明の課題は、そのようなデバッグ方法を実現させるためのプログラムを記録した記録媒体を提供することにある。

【0017】**【課題を解決するための手段】**

上記課題を解決するため、この発明のデバッグ装置は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段とを備えたことを特徴とする。

【0018】

この発明のデバッグ装置では、逐次化手段は、並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、記憶手段が上記デバッグ情報を記憶する。したがって、デバッグを行う作業者は、上記記憶手段の記憶内容、つまり上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0019】

また、別の面では、この発明のデバッグ装置は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とする。

【0020】

この発明のデバッグ装置では、記憶手段は、デバッグ対象となった並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶している。変換手段は、上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0021】

また、別の面では、この発明のデバッグ装置は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、

並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する逐次化手段と、

上記デバッグ情報を記憶する記憶手段と、

上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する変換手段を備えたことを特徴とする。

【0022】

この発明のデバッグ装置では、逐次化手段は、並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、記憶手段が上記デバッグ情報を記憶する。変換手段は、上記デバッグ情報に基づいて、上記並列プログラムと逐次プログラムとの間で対応する情報同士を変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0023】

上記並列プログラムでの情報を表示する一方、その情報に対応した上記逐次プログラムでの情報を隠すのが望ましい。そのようにした場合、作業者は上記変換手段を介してあたかも上記並列プログラムに対して直接デバッグを行っているかのような感覚でデバッグ作業を行うことができる。例えば、作業者は元の並列プログラム内の行番号や変数名を使って不具合箇所を特定することができる。そして、元の並列プログラム内の行番号や変数名を使ってその不具合を修正して解消することができる。

【0024】

一実施形態のデバッグ装置は、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応を表すことを特徴とする。

【0025】

この一実施形態のデバッグ装置では、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応を表す。したがって、デバッグを行う作業者は、上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で行番号の対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。特に、変換手段が上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で対応する行番号同士を相

互に変換する場合は、作業者はデバッグ作業をさらに効率良く短時間で行うことができる。

【0026】

一実施形態のデバッグ装置は、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の変数名の対応を表すことを特徴とする。

【0027】

この一実施形態のデバッグ装置では、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の変数名の対応を表す。したがって、デバッグを行う作業者は、上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で変数名の対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。特に、変換手段が上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間に対応する変数名同士を相互に変換する場合は、作業者はデバッグ作業をさらに効率良く短時間で行うことができる。

【0028】

一実施形態のデバッグ装置は、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応と変数名の対応とを表すことを特徴とする。

【0029】

この一実施形態のデバッグ装置では、上記デバッグ情報は上記並列プログラムと逐次プログラムとの間の行番号の対応と変数名の対応とを表す。したがって、デバッグを行う作業者は、上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間で行番号の対応と変数名の対応とを容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。特に、変換手段が上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの間に対応する行番号同士、変数名同士を相互に変換する場合は、作業者はデバッグ作業をさらに効率良く短時間で行うことができる。

【0030】

この発明のデバッグ方法は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、上記デバッグ情報を記憶手段に記憶させることを特徴とする。

【0031】

この発明のデバッグ方法では、逐次化手段によって並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、上記デバッグ情報を記憶手段に記憶させる。したがって、デバッグを行う作業者は、上記記憶手段の記憶内容、つまり上記デバッグ情報に基づいて上記並列プログラムと逐次プログラムとの対応を容易に取ることができる。これにより、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0032】

また、別の面では、この発明のデバッグ方法は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とする。

【0033】

この発明のデバッグ方法では、デバッグ対象となった並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を記憶手段に記憶させておく。そして、上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0034】

また、別の面では、この発明のデバッグ方法は、並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ方法であって、

逐次化手段によって並列プログラムを逐次プログラムに変換するとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成し、上記デバッグ情報を記憶手段に記憶させ、

上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換することを特徴とする。

【0035】

この発明のデバッグ方法では、逐次化手段によって並列プログラムを逐次プログラムに変換する。それとともに、上記並列プログラムと逐次プログラムとの対応関係を表すデバッグ情報を生成する。そして、上記デバッグ情報を記憶手段に記憶させる。上記デバッグ情報に基づいて、変換手段によって上記並列プログラムと逐次プログラムとの間で対応する情報同士を相互に変換する。したがって、デバッグを行う作業者は、上記並列プログラムと逐次プログラムとの間で対応する情報同士を自身で相互に変換する必要がない。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0036】

この発明の記録媒体は、上述のいずれか一つのデバッグ方法を実現させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体である。

【0037】

この発明の記録媒体に記録されたプログラムをコンピュータ読み取りして実行すれば、上述のデバッグ方法を実現することができる。したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0038】

【発明の実施の形態】

以下、この発明を図示の実施の形態により詳細に説明する。

【0039】

図3は、本発明の一実施形態のデバッグ装置の構成を示している。

【0040】

このデバッグ装置は、図中の並列プログラム31を逐次プログラム34に変換してデバッグを行うために、逐次化手段としての逐次化部32と、記憶手段とし

てのデータベース 35 と、逐次プログラムをデバッグするツールとしてのデバッガ 36 と、変換手段としてのデバッガ・インタフェース 37 とを備えている。

【0041】

逐次化部 32 は、逐次化処理を行って並列プログラム 31 を逐次プログラム 34 に変換する。なお、逐次化処理としては、公知の方法を用いることができる。

【0042】

逐次化部 32 の内部にはデバッグ情報生成部 33 が含まれている。デバッグ情報生成部 33 は、逐次化処理と同時に、並列プログラム 31 と逐次プログラム 34 との対応関係を表すデバッグ情報を生成する。このデバッグ情報はデータベース 35 に格納される。

【0043】

デバッガ 36 は、作業者が入力したコマンドをデバッガ・インタフェース 37 を介して受け取り、受け取ったコマンドに応じて逐次プログラム 34 をデバッグする。

【0044】

デバッガ・インタフェース 37 は、デバッグ作業 38 からコマンドを受け取り、データベース 35 を参照して、必要に応じて変換したコマンドをデバッガ 36 に対して発行する。また、デバッガ 36 からの出力を受け取り、データベース 35 を参照して、必要に応じて変換した出力をデバッグ作業 38 に向けて出力する。

【0045】

なお、このデバッグ装置は、上述の構成要素以外に、デバッグ対象となったプログラム表示し得る表示装置や、デバッグ作業の出力を作業者に提供し得る出力装置を備えている。

【0046】

次に、上記の様に構成されたデバッグ装置によってデバッグを行う処理を説明する。

【0047】

- i) まず、並列プログラム 31 として図 4 に示すような内容を持つものが、

逐次化部 3 2 に入力される。

【0048】

ここで、図 4 中の左欄に示す行番号 (10, 11, ..., 29) 45 は、説明のために付加したものであり、並列プログラム 3 1 には含まれない。

【0049】

この並列プログラム 3 1 は、C 言語を拡張した文法をもっており、キーワード “p a r” の後の中括弧 `{ }` で囲まれた範囲内に、並列に実行する処理が記述されている。この例では、点線で囲まれた部分の処理 4 1、4 2 は並列に実行される処理である。これらの処理 4 1、4 2 は、互いに独立した処理であるため、互いに同じ名前の変数 `i` や変数 `k` などを含んでいても構わない。理解の容易のために、処理 4 1 に含まれた変数 `i` に符号 4 3、処理 4 2 に含まれた変数 `i` に符号 4 4 を付している。

【0050】

ii) 上記並列プログラム 3 1 を逐次化部 3 2 が C 言語を用いて逐次化して、逐次プログラム 3 4 として図 5 乃至図 6 に示すような内容を持つものが出力される。

【0051】

なお、図 5 は前半部分、図 6 は後半部分に相当する。図 5、図 6 中の左欄に示す行番号 (100, 101, ..., 155) 55 は、説明のために付加したものであり、逐次プログラム 3 4 には含まれない。

【0052】

この逐次プログラム 3 4 は、点線で囲まれた部分の処理 5 1、5 2 が、図 4 中の処理 4 1、4 2 にそれぞれ対応しており、逐次動作するようになっている。この逐次プログラム 3 4 での変数 `i__0` (符号 5 3 で示す)、変数 `i__1'` (符号 5 4 で示す) は、それぞれ並列プログラム 3 1 中の処理 4 1 での変数 `i` (符号 4 3 で示す)、処理 4 2 での変数 `i` (符号 4 4 で示す) に対応している。このように、処理 4 1、4 2 に含まれた同じ変数名の後ろにそれぞれ `_0`、`_1` というようなスコープ番号を付して、それらを区別している。この理由は、逐次プログラム (の一つの処理ブロック内) では、同じ変数名は同じ実体を表すものとして

扱われ、同じ変数名であって実体が異なるものが存在することは許されないからである。

【0053】

この逐次プログラム 34 に含まれた処理 51、処理 52 は、図 5 中の点線で囲まれた部分以外の処理にしたがって、疑似的に並列に実行される。

【0054】

iii) 逐次化部 32 が並列プログラム 31 を逐次プログラム 34 へ変換すると同時に、逐次化部 32 内部のデバッグ情報生成部 33 が、並列プログラム 31 と逐次プログラム 34 との対応関係を表すデバッグ情報を生成する。生成されたデバッグ情報はデータベース 35 に格納される。

【0055】

この例では、デバッグ情報は、図 7 に示す行番号対応情報 61 と、図 8 に示す変数名対応情報 62 と、図 9 に示すスコープ情報 63 とを含んでいる。

【0056】

図 7 に示す行番号対応情報 61 は、並列プログラム 31 で或る処理を行う行番号（右欄 65 に並べて示す）と、逐次プログラム 34 でその処理と同じ処理を行う行番号（左欄 64 に並べて示す）との対応関係を表している。例えば、並列プログラム（“prog__par. c”）の 15 行目は、逐次プログラム（“prog__seq. c”）の 117 行目に対応することを示す。この行番号対応情報 61 は、逐次化部 32 が並列プログラム 31 内の行を逐次プログラム 34 の行に変換する際に、デバッグ情報生成部 33 が、元の行の並列プログラム 31 内での行番号と変換先の逐次プログラム 34 内での行番号とを関連付けてデータベース 35 に登録することにより、生成される。

【0057】

図 8 に示す変数名対応情報 62 は、並列プログラム 31 内での或るデータを保持する変数名（右欄 67 に並べて示す）と、逐次プログラム 34 内でのそのデータと同じデータを保持する変数名（左欄 66 に並べて示す）との対応関係を表している。例えば、並列プログラム 31 での変数 i は、逐次プログラム 34 での変数 i__0 と変数 i__1 に対応することを示す。また、並列プログラム 31 での変

数 k は、逐次プログラム 34 での変数 k_0 と変数 k_1 に対応することを示す。この変数名対応情報 62 は、逐次化部 32 が並列プログラム 31 での変数名を逐次プログラム 34 での変数名に変換する際に、デバッグ情報生成部 33 が、並列プログラム 31 での変数名と逐次プログラム 34 での変数名とを関連付けてデータベース 35 に登録することにより、生成される。

【0058】

図 9 に示すスコープ情報 63 は、並列プログラム 31 内での変数の有効範囲（始行と終行との二つの行番号で指定される。左欄 68 に並べて示す）とスコープ番号（右欄 69 に並べて示す）との対応関係を表している。スコープ番号は、並列プログラム 31 での変数の有効範囲をそれぞれ区別するために、逐次化処理に際して導入される番号である。例えば、並列プログラム（“prog_par.c”）の 13 行目から 18 行目までの範囲は、スコープ番号 0 に対応することを示す。また、並列プログラム（“prog_par.c”）の 21 行目から 26 行目までの範囲は、スコープ番号 1 に対応することを示す。このスコープ情報 63 は、逐次化部 32 が入力 of 並列プログラム 31 の構造を解析する際に、デバッグ情報生成部 33 が、並列プログラム 31 内の変数の有効範囲（二つの行番号）とその範囲に対応したスコープ番号とを関連付けてデータベース 35 に登録することによって、生成される。

【0059】

iv) 上述のようなデバッグ情報がデータベース 35 に格納された後、作業者がデバッガ・インタフェース 37 を介してデバッガ 36 を動作させて、デバッグを行う。

【0060】

図 10 は、デバッグ作業者がコマンドを発行したときの処理の流れを示している。

【0061】

まず、作業者 38 は、デバッガ・インタフェース 37 に、デバッグを行うためのコマンドを発行する。

【0062】

デバッガ・インタフェース 37 は、データベース 35 内の行番号対応情報 61 (図 7 に示す) を参照して、そのコマンドに並列プログラム 31 での行番号が含まれているか否かを調べる (ステップ S1)。

【0063】

そのコマンドに並列プログラム 31 での行番号が含まれている場合はステップ S2 へ進む一方、含まれていない場合はステップ S3 へ進む。

【0064】

ステップ S2 では、データベース 35 内の行番号対応情報 61 を参照した結果に基づいて、並列プログラム 31 での行番号を逐次プログラム 34 での対応する行番号に変換する。そして、ステップ S3 へ進む。

【0065】

ステップ S3 では、データベース 35 内の変数名対応情報 62 (図 8 に示す) を参照して、そのコマンドに並列プログラム 31 での変数名が含まれているか否かを調べる。

【0066】

そのコマンドに並列プログラム 31 での変数名が含まれている場合はステップ S4 へ進む一方、含まれていない場合はステップ S9 へ進む。

【0067】

ステップ S4 では、データベース 35 内の変数名対応情報 62 を参照した結果に基づいて、並列プログラム 31 での変数名が逐次プログラム 34 での複数の変数名に対応しているか否かを調べる。

【0068】

並列プログラム 31 での変数名が逐次プログラム 34 での複数の変数名に対応している場合はステップ S5 - S7 へ進む一方、複数の変数名に対応していない場合、つまり 1 つの変数名のみに対応している場合はステップ S8 へ進む。

【0069】

ステップ S5 - S7 では、並列プログラム 31 での変数名が逐次プログラム 34 での複数の変数名に対応している場合であるから、複数の変数名をそれぞれ区別するためにスコープ番号を利用する。具体的には、まずステップ S5 では、現

在デバッグを行っているプログラム上の位置（これを「デバッグ位置」と呼ぶ。）を調べる。続いて、ステップS6では、データベース35内のスコープ情報63（図9に示す）を参照して、ステップS5で取得した現在のデバッグ位置を含む範囲に対応するスコープ番号を得る。続いて、ステップS7では、ステップS6で得たスコープ番号を利用して、並列プログラム31での変数名を逐次プログラム34での対応する変数名に変換する。例えば、ステップS6で得たスコープ番号が0であれば、並列プログラム31での変数*i*は逐次プログラム34での変数*i*__0に変換される。また、ステップS6で得たスコープ番号が1であれば、並列プログラム31での変数*i*は逐次プログラム34での変数*i*__1に変換される。この後、ステップS9へ進む。

【0070】

ステップS8では、並列プログラム31での変数名が逐次プログラム34での1つの変数名のみに対応している場合であるから、スコープ番号を用いる必要はない。そこで、単に、データベース35内の変数名対応情報62を参照した結果に基づいて、並列プログラム31での変数名を逐次プログラム34での対応する変数名に変換する。そして、ステップ9へ進む。

【0071】

ステップS9では、上述のステップS1－S8の処理を経て変換されたコマンドをデバッガ36に対して発行する。ただし、コマンドが行番号と変数名とのいずれも含まない場合（ステップS1，S3でいずれもNOの場合）は、コマンドは変換されずにそのまま発行される。

【0072】

図11は、デバッガ36からの出力に対する処理の流れを示している。

【0073】

デバッガ36からの出力を受けると、まず、デバッガ・インタフェース37は、データベース35内の行番号対応情報61（図7に示す）を参照して、その出力に逐次プログラム34での行番号が含まれているか否かを調べる（ステップS11）。

【0074】

その出力に逐次プログラム 3 4 での行番号が含まれている場合はステップ S 1 2 へ進む一方、含まれていない場合はステップ S 1 3 へ進む。

【0 0 7 5】

ステップ S 1 2 では、データベース 3 5 内の行番号対応情報 6 1 を参照した結果に基づいて、逐次プログラム 3 4 での行番号を並列プログラム 3 1 での対応する行番号に変換する。そして、ステップ S 1 3 へ進む。

【0 0 7 6】

ステップ S 1 3 では、データベース 3 5 内の変数名対応情報 6 2（図 8 に示す）を参照して、その出力に逐次プログラム 3 4 での変数名が含まれているか否かを調べる。

【0 0 7 7】

その出力に逐次プログラム 3 4 での変数名が含まれている場合はステップ S 1 4 へ進む一方、含まれていない場合はステップ S 1 5 へ進む。

【0 0 7 8】

ステップ S 1 4 では、データベース 3 5 内の変数名対応情報 6 2 を参照した結果に基づいて、逐次プログラム 3 4 での変数名を並列プログラム 3 1 での対応する変数名に変換する。そして、ステップ 1 5 へ進む。

【0 0 7 9】

ステップ S 1 5 では、上述のステップ S 1 1 - S 1 4 の処理を経て変換された出力を作業者に対して提供する。ただし、出力が行番号と変数名とのいずれも含まない場合（ステップ S 1 1, S 1 3 でいずれも NO の場合）は、出力は変換されずにそのまま提供される。

【0 0 8 0】

上述のようにしてデバッグを行うようにした場合、作業者は、並列プログラム 3 1 と逐次プログラム 3 4 との間で対応する行番号同士、変数名同士を自身で相互に変換する必要がない。

【0 0 8 1】

さらに、表示装置で並列プログラム 3 1 での情報を表示する一方、その情報に対応した逐次プログラム 3 4 での情報を隠すようにした場合、作業者はデバッグ

・ インターフェース 37 を介してあたかも並列プログラム 31 に対して直接デバッグを行っているかのような感覚でデバッグ作業を行うことができる。例えば、作業者は元の並列プログラム 31 内の行番号や変数名を使って不具合箇所を特定することができる。そして、元の並列プログラム 31 内の行番号や変数名を使ってその不具合を修正して解消することができる。

【0082】

したがって、作業者はデバッグ作業を効率良く短時間で行うことができる。

【0083】

なお、このデバッグ装置は、逐次プログラムを実行可能な一般的な電子計算機上で、上述のデバッグを行う処理を実行するためのプログラム（ソフトウェア）を用いて構成され得る。そのプログラムは、電子計算機が備える外部記憶装置（固定記憶装置）に記憶されていても良い。また、記録媒体（コンパクトディスク等）に書き込まれたそのようなプログラムを、読み取り可能なドライブで読み取るようになっていても良い。

【0084】

【発明の効果】

以上より明らかなように、この発明のデバッグ装置およびデバッグ方法によれば、作業者がデバッグ作業を効率良く行うことができる。

【0085】

また、この発明の記録媒体によれば、そのようなデバッグ方法を実現させることができる。

【図面の簡単な説明】

【図 1】 並列プログラムを逐次プログラムに変換してデバッグを行う従来の方法を説明する図である。

【図 2】 元の並列プログラムではなく逐次プログラムを修正し、修正後の逐次プログラムを並列プログラムに変換する従来の方法を説明する図である。

【図 3】 本発明の一実施形態のデバッグ装置の構成を示す図である。

【図 4】 並列プログラムの例を示す図である。

【図 5】 図 4 の並列プログラムを変換して得られた逐次プログラム（前半

部分)を示す図である。

【図6】 図4の並列プログラムを変換して得られた逐次プログラム(後半部分)を示す図である。

【図7】 データベースに格納されたデバッグ情報としての行番号対応情報を例示する図である。

【図8】 データベースに格納されたデバッグ情報としての変数名対応情報を例示する図である。

【図9】 データベースに格納されたデバッグ情報としてのスコープ情報例示する図である。

【図10】 デバッグを行う処理(コマンド発行時)のフローを示す図である。

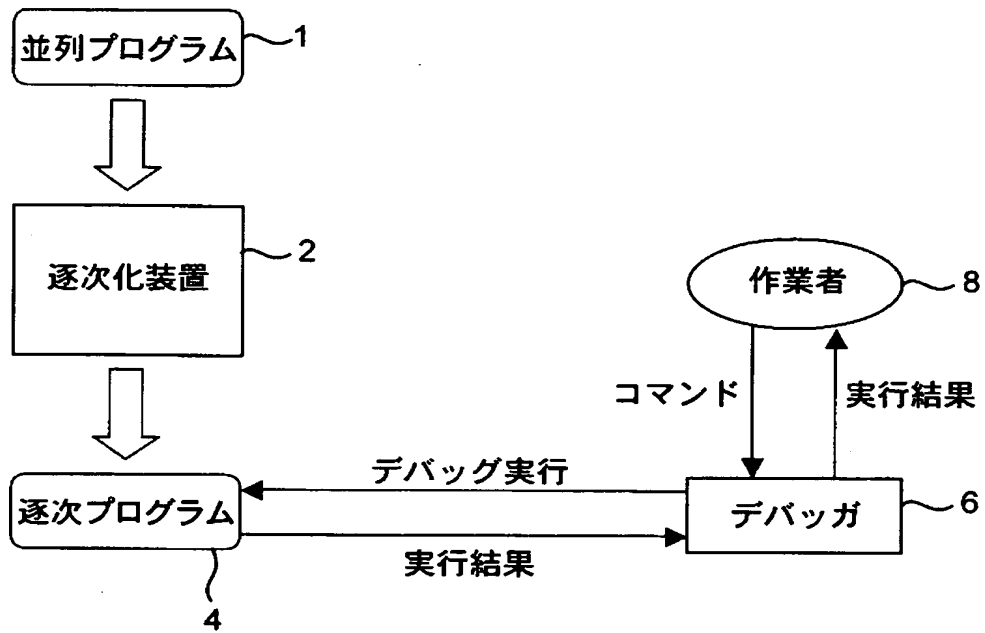
【図11】 デバッグを行う処理(結果出力時)のフローを示す図である。

【符号の説明】

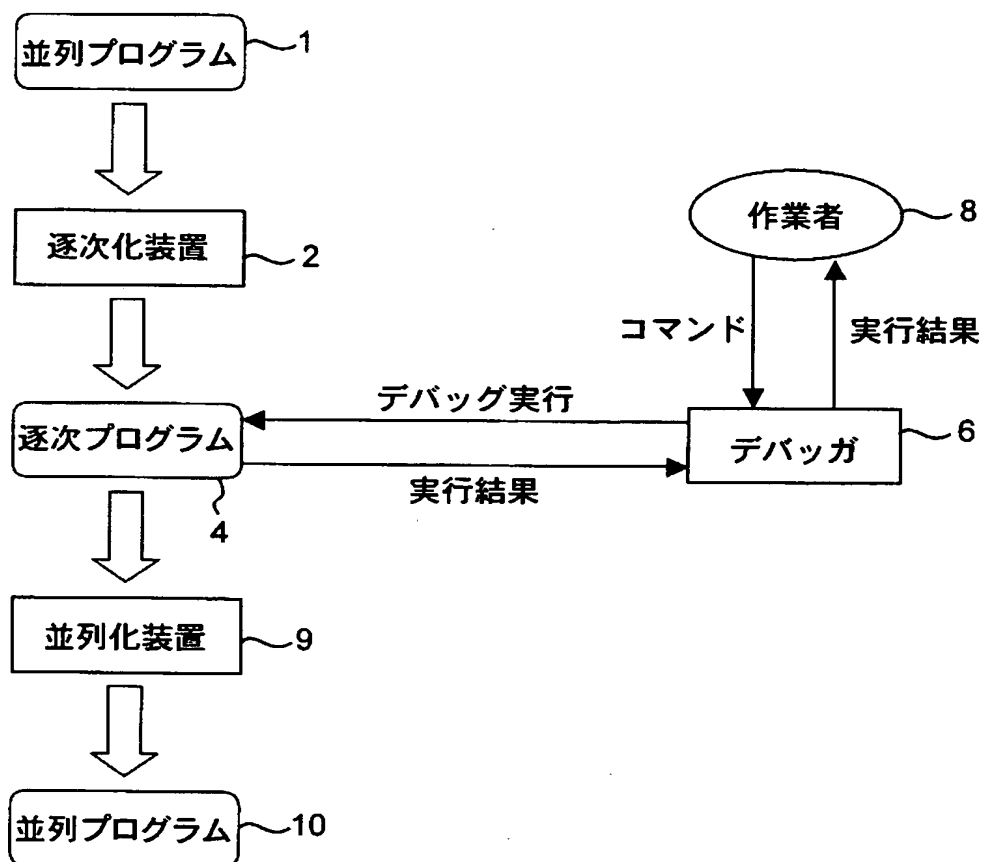
- 31 並列プログラム
- 32 逐次化部
- 33 デバッグ情報生成部
- 34 逐次プログラム
- 35 データベース
- 36 デバッガ
- 37 デバッガ・インタフェース
- 61 行番号対応情報
- 62 変数名対応情報
- 63 スコープ情報

【書類名】 図面

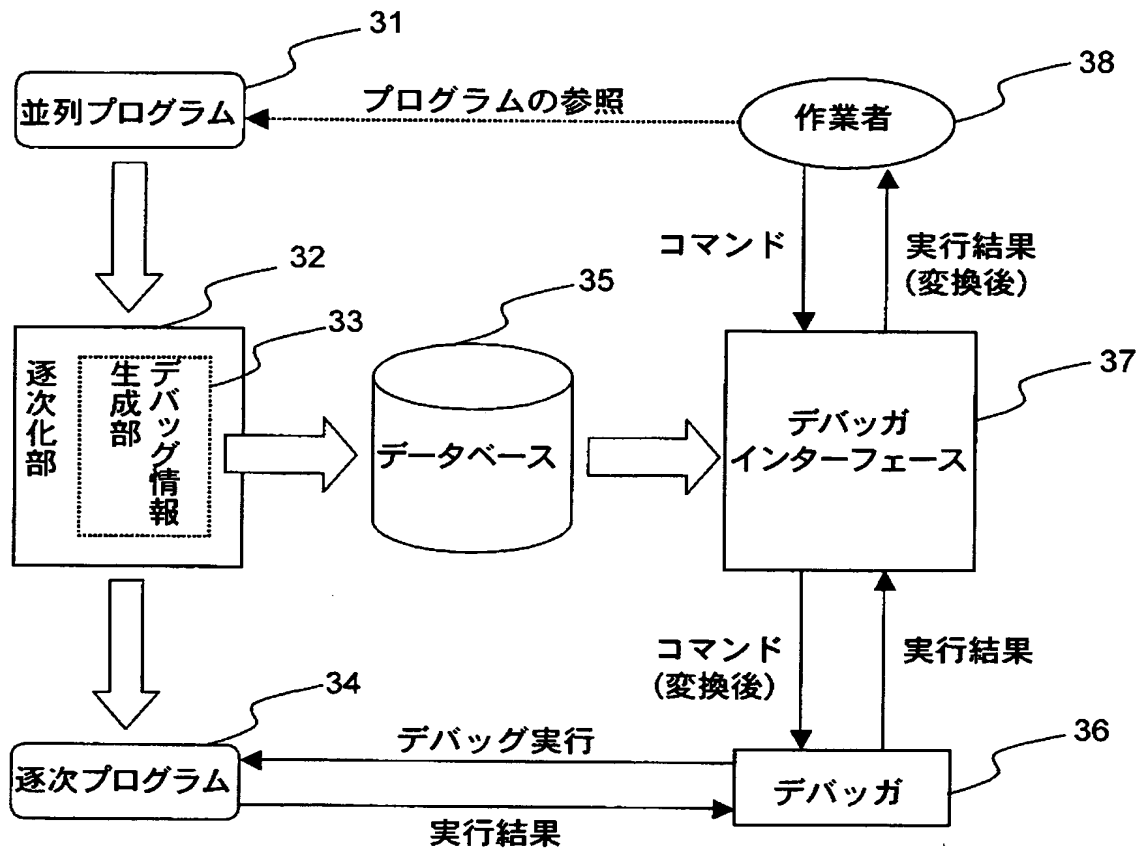
【図 1】



【図 2】



【図 3】



【図 4】

The diagram shows a C-like code snippet with line numbers 10 to 29 on the left. The code is enclosed in a large box labeled 45. Inside this box, there are two parallel blocks separated by a horizontal line. The top block is labeled 41 and contains lines 12-19. The bottom block is labeled 42 and contains lines 20-27. A label 43 points to the 'par {' line, and a label 44 points to the 'int i, k, sum_k=0;' line in the second block. The code calculates the sum of squares and the sum of products, then adds them to result3.

```
10 int result1, result2, result3;
11 par {
12     {
13         int i, k, sum_k=0;
14         for (i=1; i<=10; i++) {
15             k=i*2;
16             sum_k=sum_k+k;
17         }
18         result1=sum_k;
19     }
20     {
21         int i, k, sum_k=0;
22         for (i=1; i<=10; i++) {
23             k=i*i;
24             sum_k=sum_k+k;
25         }
26         result2=sum_k;
27     }
28 }
29 result3=result1+result2;
```

【図 5】

```
100  int main_result1, main_result2, main_result3;
101  int i_0, k_0, sum_k_0=0;
102  int i_1, k_1, sum_k_1=0;
103  int thread=THREAD_0;
104  int state_0=STATE_0_0;
105  int state_1=STATE_1_0;
106
107  while(!(state_0==FINISHED && state_1==FINISHED)) {
108      switch (thread) {
109          case THREAD_0:
110              switch (state_0) {
111                  case STATE_0_0:
112                      i_0=1;
113                      state_0=STATE_0_1;
114                      break;
115
116                  case STATE_0_1:
117                      k_0=i_0*2;
118                      sum_k_0=sum_k_0+k_0;
119                      i_0=i_0+1;
120                      if (!(i_0<=10))
121                          state_0=STATE_0_2;
122                      break;
123
124                  case STATE_0_2:
125                      main_result1=sum_k_0;
126                      state_0=FINISHED;
127              }
128              thread=THREAD_1;
129              break;
```

Diagram annotations:

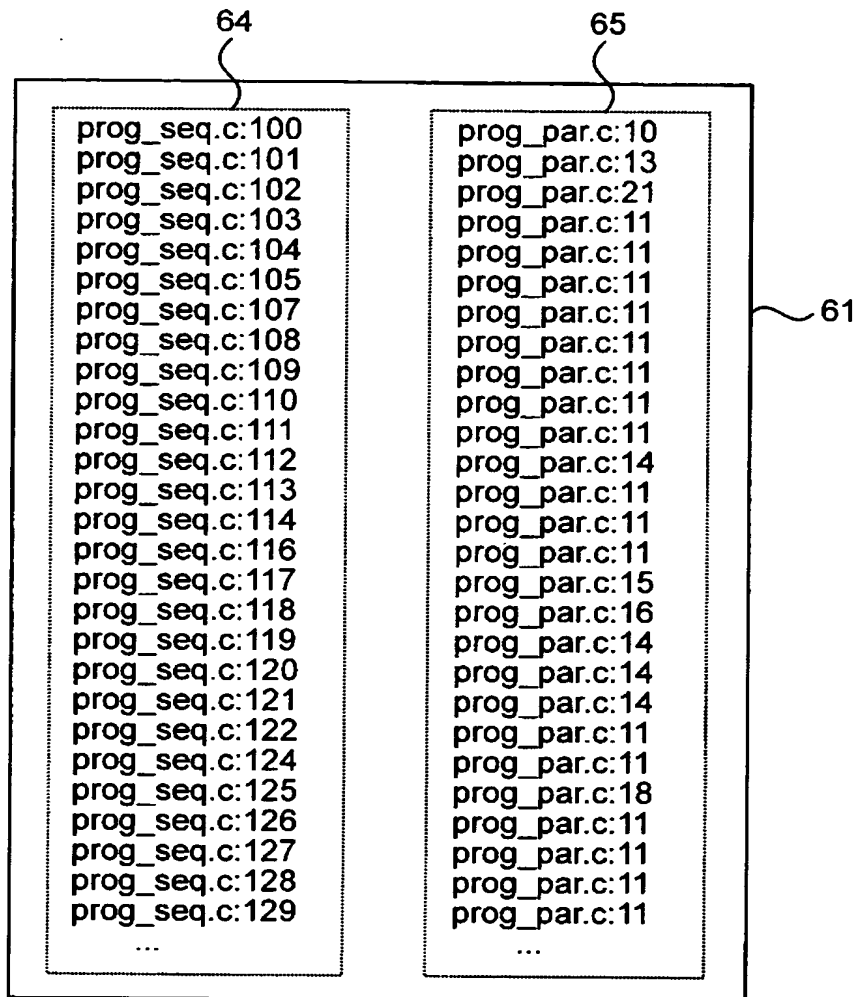
- 55: points to line 100.
- 54: points to line 101.
- 53: points to line 102.
- 51: points to the nested switch statement in the while loop.

【図 6】

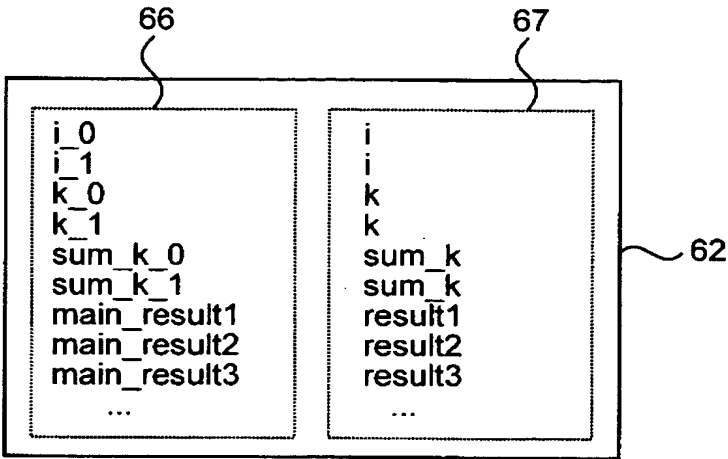
```
130
131 case THREAD_1:
132     switch (state_1) {
133     case STATE_1_0:
134         i_1=1;
135         state_1=STATE_1_1;
136         break;
137
138     case STATE_1_1:
139         k_1=i_1*i_1;
140         sum_k_1=sum_k_1+k_1;
141         i_1=i_1+1;
142         if (!(i_1<=10))
143             state_1=STATE_1_2;
144         break;
145
146     case STATE_1_2:
147         main_result2=sum_k_1;
148         state_1=FINISHED;
149     }
150     thread=THREAD_0;
151     break;
152 }
153 }
154
155 main_result3=main_result1+main_result2;
```

52

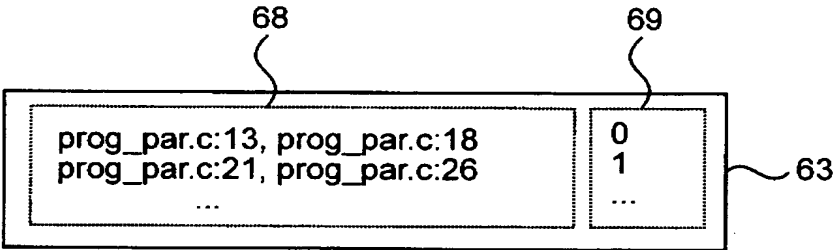
【図 7】



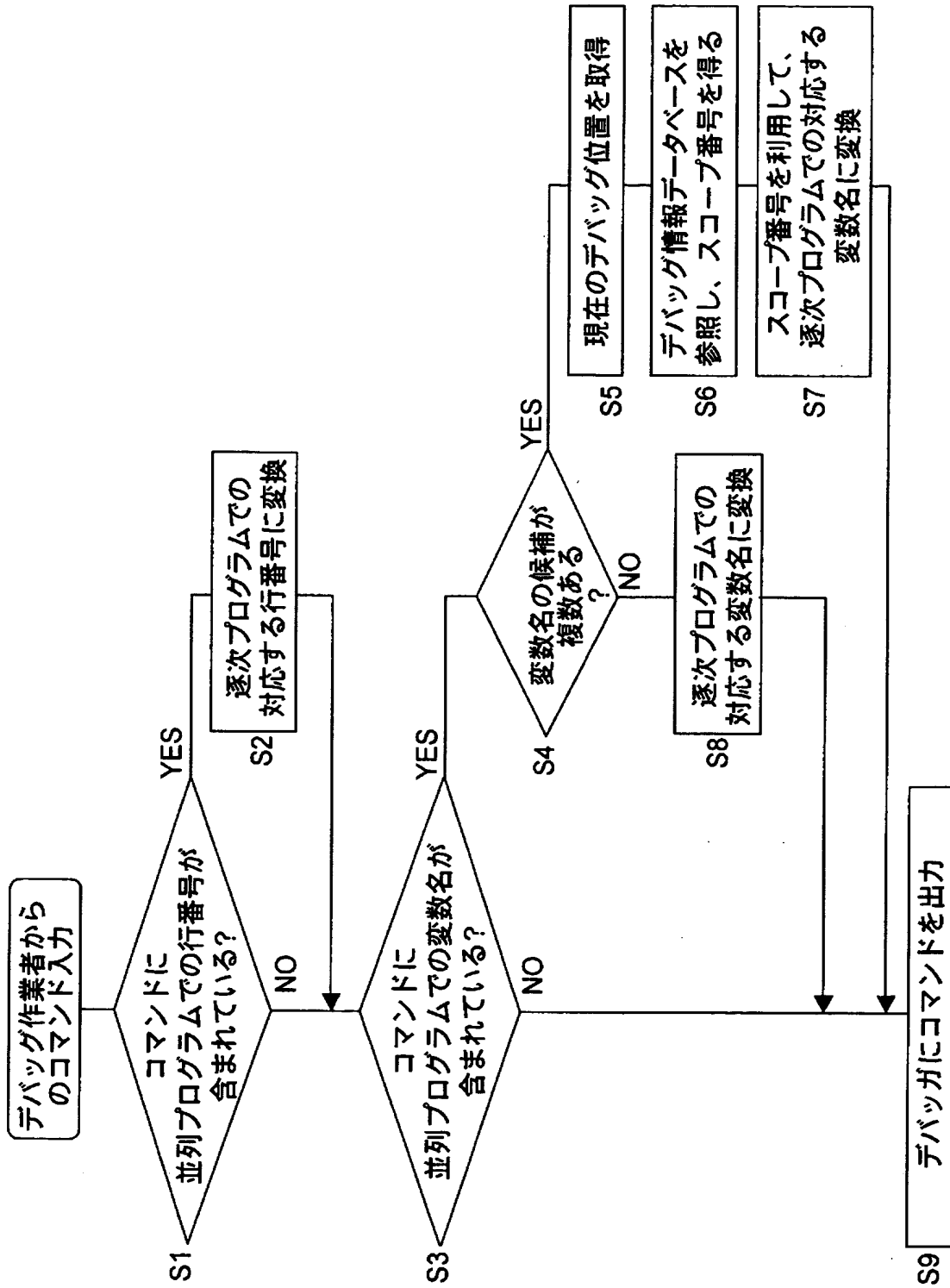
【図 8】



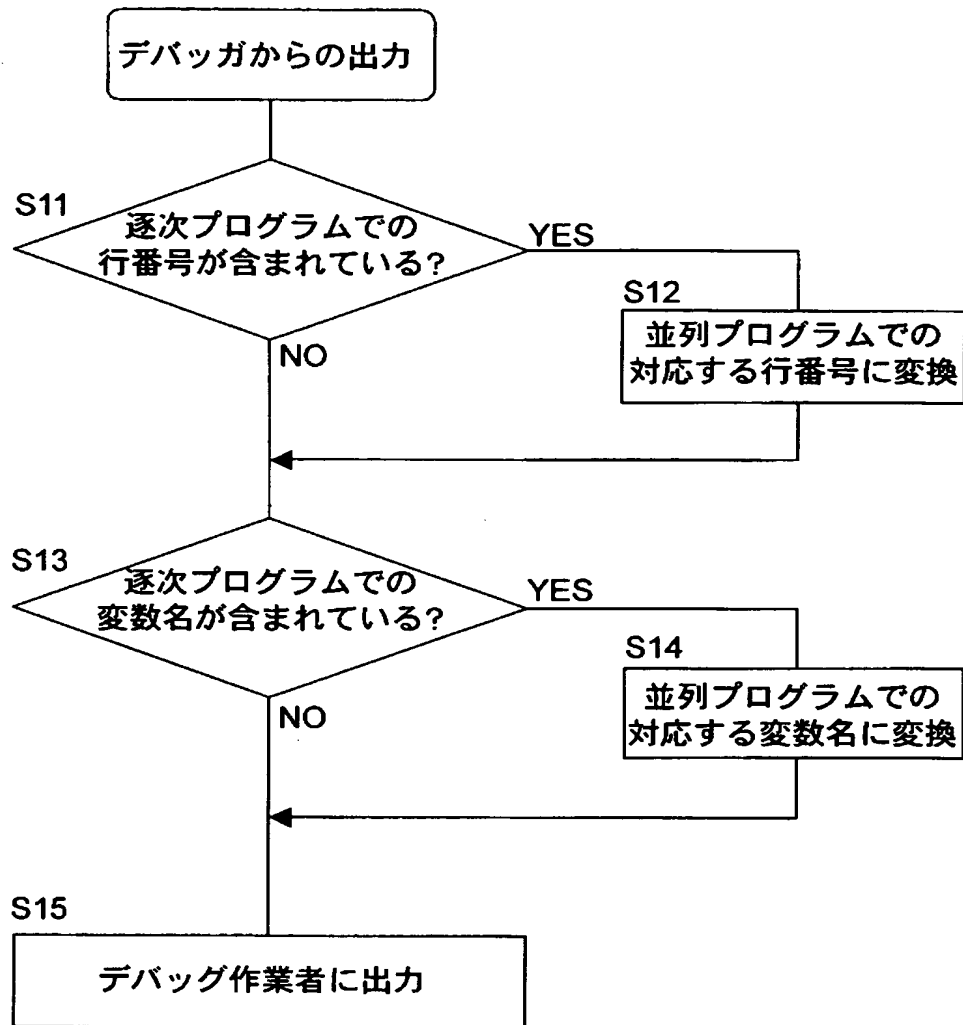
【図 9】



【図 10】



【図 11】



【書類名】 要約書**【要約】**

【課題】 並列プログラムを逐次プログラムに変換してデバッグを行うのに用いられるデバッグ装置であって、作業者がデバッグ作業を効率良く行うことができるものを提供すること。

【解決手段】 並列プログラム 31 を逐次プログラム 34 に変換するとともに、並列プログラム 31 と逐次プログラム 34 との対応関係を表すデバッグ情報を生成する逐次化手段 32 を備える。そのデバッグ情報を記憶する記憶手段 35 を備える。そのデバッグ情報に基づいて、並列プログラム 31 と逐次プログラム 34 との間で対応する情報同士を相互に変換する変換手段 37 を備える。

【選択図】 図 3

特願 2 0 0 2 - 3 5 6 9 7 5

出 願 人 履 歴 情 報

識別番号

[0 0 0 0 0 5 0 4 9]

1. 変更年月日

1 9 9 0 年 8 月 2 9 日

[変更理由]

新規登録

住 所

大阪府大阪市阿倍野区長池町 2 2 番 2 2 号

氏 名

シャープ株式会社